Praktische Prüfung

zum

Mathematisch-Technischen Softwareentwickler

**Dokumentation  
Erstellung eines Zufallgenerators**

***Dokumentenvorlage für wissenschaftliche Arbeiten***

***Für private Zwecke frei verwendbar***

***Schriftarten: Arial (Überschriften)/Times New Roman (Fließtext)***

***Seitenzählung römische/arabische Ziffern***

Autor: **Bjarne Herrmann**  
 Leimkaul 4  
 51143-Köln  
 T.: 0176 / 64484501  
 E-Mail: herrmann.bjarne@outlook.de

Ident-Nr.: 142/5002317989

Ausbildungs-Nr.: 142/18748

# Inhaltsverzeichnis

[Kurzfassung Fehler! Textmarke nicht definiert.](#_Toc504921067)

[Inhaltsverzeichnis II](#_Toc504921068)

[Abbildungsverzeichnis II](#_Toc504921069)

[Tabellenverzeichnis II](#_Toc504921070)

[Abkürzungsverzeichnis Fehler! Textmarke nicht definiert.](#_Toc504921071)

[Vorwort Fehler! Textmarke nicht definiert.](#_Toc504921072)

[1 Einleitung Fehler! Textmarke nicht definiert.](#_Toc504921073)

[Textbausteine/Formulierungsvorschläge **Fehler! Textmarke nicht definiert.**](#_Toc504921074)

[2 Musterseite (Ü 1) 2](#_Toc504921075)

[2.1 Überschrift 2 2](#_Toc504921076)

[2.1.1 Überschrift 3 2](#_Toc504921077)

[2.1.2 Überschrift 3 2](#_Toc504921078)

[2.1.2.1 Überschrift 4 2](#_Toc504921079)

[2.1.2.2 Überschrift 4 2](#_Toc504921080)

[3 Hauptteil 1: Status Quo 2](#_Toc504921081)

[3.1 Stand der Wissenschaft 2](#_Toc504921082)

[3.2 Historische Einordnung 2](#_Toc504921083)

[3.3 Quellenlage 2](#_Toc504921084)

[4 Hauptteil 2: Grundlagen und Methodik 2](#_Toc504921085)

[4.1 Theoretischer Bezugsrahmen 2](#_Toc504921086)

[4.2 Methodische Vorgehensweise 2](#_Toc504921087)

[5 Hauptteil 3: Empirische Untersuchung 2](#_Toc504921088)

[5.1 Analysen 2](#_Toc504921089)

[5.2 Interpretation der Ergebnisse 2](#_Toc504921090)

[6 Zusammenfassung 2](#_Toc504921091)

[Literaturverzeichnis II](#_Toc504921092)

[Anhänge II](#_Toc504921093)

[Anhang 1 II](#_Toc504921094)

[Anhang 2 II](#_Toc504921095)

[Eidesstattliche Versicherung II](#_Toc504921096)

[Stichwortverzeichnis II](#_Toc504921097)

# Abbildungsverzeichnis

Abbildung 1: Korrektureule 2

# Tabellenverzeichnis

Tabelle 1: Beispieltabelle mit einer besonders langen Tabellenüberschrift, um den Umbruch des Textes darzustellen 2

# Aufgabenbeschreibung

Thema ist die Generierung und Bewertung von Zufallszahlen mithilfe diverser Zufallszahlengeneratoren. Jeder Generator besitzt einen individuellen Algorithmus für die Bestimmung solcher Ergebnisse.

In der Stochastik sind folgende Eigenschaften für Zufallsgenerierungen relevant:

* Determiniertheit
* Generatorklasse
  + Linear-Kongruenz-Generator (englisch: LCG)
  + Standardnormalverteilte Erzeugung mittels der Polar-Methode
* Verteilungseigenschaften
  + Stetig / Diskret
  + Verteilungsfunktion
  + Wertebereich
  + Verteilungsparameter
  + Gütekriterium
    - Autokorrelation
    - Sequenz-Up-Down

Grund der Thematik ist die Wichtigkeit von Zufallszahlen für verschiedene Kontexte, wie zum Beispiel Simulationen, Randomisierung, Stochastische Aussagen etc.

## Linearer-Kongruenz-Generator

Für den Linearen-Kongruenz-Generator (entworfen 1949 von D.H. Lehmer) wird die Modulo-Berechnung mit diversen Parametern verwendet. Mit Hilfe dieser Parameter wird iterativ, basierend auf der jeweils zuvor erzeugten Zahl die nächste Zufallszahl erzeugt. Dabei gilt:

* xi+1 = (a \* xi + c) mod m

Mit:

* m > 0
* 0 ≤ a < m
* 0 ≤ c < m
* 0 ≤ x0 < m

Zudem ist zu beachten, dass unter dem Einhalten bestimmter Bedingungen die Periodenlänge (Anzahl an Zahlen, bis eine Wiederholung vorkommt) maximiert werden kann:

* c mod m = 1
* Ist p ein Primteiler von m ⬄ Ist p Teiler von a-1
* Ist 4 Teiler von m ⬄ 4 Teiler von a-1

Folgend ist eine Auflistung bekannter Parameterwahlen für den LCG:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Name* | *Modul m* | *Multiplikator a* | *Inkrement c* | *Startwert x0* |
| ANSI-C | 231 | 1103515245 | 12345 | 12345 |
| Minimal Standard | 231-1 | 16807 | 0 | 1 |
| RANDU | 231 | 65539 | 0 | 1 |
| SIMSCRIPT | 231-1 | 630360016 | 0 | 1 |
| NAG’s LCG | 259 | 1313 | 0 | 123456789 |
| Maple’s LCG | 1012-11 | 427419669081 | 0 | 1 |

## Polar-Methode

Die Polar-Methode (von George Marsaglia und Thomas A. Bray) basiert auf der Berechnung mittels gleichverteilter Zufallszahlen, welche in Polarkoordinaten umgewandelt werden. Der Algorithmus für die Erzeugung zweier Zufallszahlen beinhaltet folgenden prozessualen Ablauf:

1. Erzeuge zwei gleichverteilte Zahlen u, v im Intervall I[-1;1]
2. Berechne q = u2 + v2 
   1. Falls q = 0 oder q ≥ 1 🡪 Beginne bei Schritt 1
3. Berechne p =
4. Berechne x1 = u \* p und x2 = v \* p

Die Zufallszahlen sind somit unabhängig und standardnormalverteilt.

## Serielle Autokorrelation

Mit der seriellen Autokorrelation kann die Güte einer Zufallszahlenfolge hinsichtlich der Abhängigkeit aufeinanderfolgender Zahlen gemessen werden. Das Verfahren prüft die Abhängigkeit mit folgender Korrelationsformel:

* ρk = **/**

Dabei stellt ρk den Korrelationswert k. Ordnung dar.

Der optimalste Wert liegt bei ρk = 0. Je weiter ρk abweicht, desto höher ist die Abhängigkeit der Zufallszahlen mit einem Abstand von k – auch „Lag k“ genannt.

## Sequenz-Up-Down-Test

Der Sequenz-Up-Down-Test vergleicht die Abhängigkeit von Zufallszahlen hinsichtlich deren aufeinanderfolgenden Größenänderung:

* Vergleiche nacheinander die Zahlen und notiere eine Bitfolge basierend auf:
  + 1 für xi < xi+1
  + 0 für xi > xi+1
* Zähle die Länge gleicher hintereinander folgenden Zahlen und notiere nacheinander die Werte als neue Zahlenfolge
* Berechne für jedes Paket:
  + k als Index für jeweiliges Paket
  + N(k) als Anzahl an Vorkommen
* Berechne für jedes k ein optimales N(k) mit:
  + N(k) =

Beispiel anhand Zahlenfolge 0, 3, 2, 13, 4, 7, 6, 1, 8, 11, 10, 5, 12, 15, 14, 9, 0

* Bitfolge: 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0
* Zählergebnis: 1, 1, 1, 1, 1, 2, 2, 2, 2, 3
* Pakete N(k):
  + k=1 🡪 N(1) = 5; k=2 🡪 N(2) = 4; k=3 🡪 N(3) = 1
* Optimales N(k):
  + k=1 🡪 N(1) = 6.75; k=2 🡪 N(2) = 2.75; k=3 🡪 N(3) = 0.58

## BjarnscheZufallsmethode

Im Rahmen der Aufgabenstellung gilt es einen eigenen Zufallsgenerator zu entwickeln. Dieser basiert auf der Nutzung von dem aktuellen Datum in Form von Tagen mit übergebenen Modulo-Wert m sowie einem Startwert x0. Es mithilfe interner Java-Bibliotheken das aktuelle Datum in Form einer Ganzzahl konvertiert und für folgende iterative Berechnung genutzt:

* counter := 0
* date := Datum in Form von Ganzzahl
* Wenn counter ungerade:
  + xn+1 = (date - xn) mod (m + counter)
* Wenn counter gerade:
  + xn+1 = (date + xn) mod (m – counter)
* counter += 1
  + Wenn (counter – 1) = 0 🡪 counter += 1
    - Da mod 0 nicht möglich ist, muss dies nach jeder Erhöhung vom counter überprüft werden

## BjarnscheGütefunktion

Für die eigene Gütefunktion wird die Veränderungsrate des Betrags des Abstands zwischen den Zahlen bewertet. Wenn dieser immer größer oder immer kleiner wird, so lässt sich dies erkennen.

Beispiel:

* Zahlenfolge: 2, 3, 4, 2, 2, 2, 1, 4, 3, 2
* Ergebnis:
  + k = 0 🡪 N(0) = 2 (zwei Teilfolgen haben keine Abstandsveränderung)
  + k = 1 🡪 N(1) = 3 (drei Teilfolgen haben Abstandsveränderung)

# Aufgabenanalyse

Aus der Beschreibung ergeben sich folgende zusammengefaste Stichpunkte / Fragestellungen, die in den nächsten Kapiteln aufgegriffen werden:

* Erzeuge Zufallszahlen mit diversen Generatoren
* Zufallszahlen sollen mit Gütekriterien Bewertet werden
* Fragestellung:
  + Wie gut sind generierte Zahlen?
  + Wann ist eine Zufallszahl per Definition „gut“?
* Es sollen basierend auf gewählte Verfahren / Parameter Zufallszahlen + Bewertung generiert werden
* Es müssen Daten eingelesen werden können, um Generierungen und Bewertungen testen zu können
* Es müssen Ergebnisse ausgegeben werden, um diese bewerten zu können

Um die Anforderung besser analysieren zu können, wird die Beschreibung in drei Bereiche gegliedert:

1. Eingabe von Informationen
2. Ausgabe von Ergebnissen für Analysen und Interpretierungen
3. Berechnung von Ergebnissen anhand eingegebenen Informationen

Die Folgenden Kapitel beschreiben jeweils die Konzeptidee. Unter Kapitel Use-Cases lassen sich detaillierte Anwendungsbeispiele aller möglichen Eingaben und resultierenden Ausgaben einsehen.

## Eingabe von Dateien

Für die Eingabe ergeben sich mehrere Use-Cases. Darunter muss zwischen folgenden unterschieden werden:

1. Der Anwender gibt einen Generator mit dazugehörigen Parametern an, mit dessen Zufallszahlen ermittelt werden soll.
2. Der Anwender besitzt bereits eine Zufallszahlenfolge und möchte mit einem Gütekriterium eine Bewertung erhalten.

Somit könnten Dateien zum Beispiel wie folgt als Text-Format eingelesen werden:

# Kommentarzeile mit Informationen zur Datei

Ziel: Zufallsgenerierung

Generator: LCG

Parameter: m=1, a=2, c=3, x0=10, n=100

Mit obigen Beispiel lässt sich eine Zufallszahlenfolge mit dem LCG-Verfahren berechnen. Für den zweiten Use-Case könnte die Eingabe wie folgt aussehen:

# Kommentarzeile mit Informationen zur Datei

Ziel: Bewertung

Zufallszahlen: 2,3,4,5,6,7,8,9

## Ausgabe von Dateien

Die Ausgabe sollte als Text-Format gegeben sein, welche zunächst den Inhalt der Eingabedatei, gefolgt von Informationen bezüglich des Ergebnisses beinhaltet. Ein Ergebnis kann von dem Algorithmus berechnet sein oder auch eine Fehlermeldung sein, die beim Einlesen aufgekommen ist. Folgend ein Beispiel:

# Kommentarzeile mit Informationen zur Datei

Ziel: Zufallsgenerierung

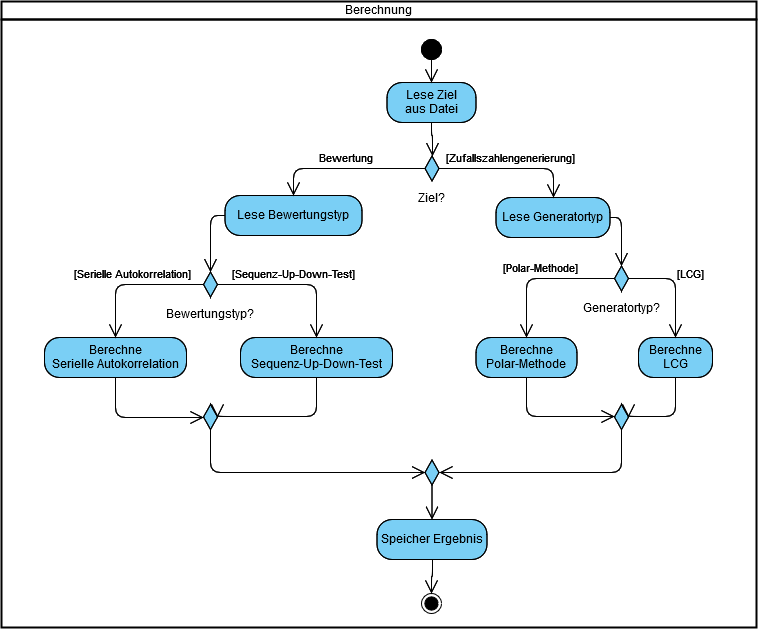
Generator: LCG

Parameter: m=1, a=2, c=3, x0=10, n=7

Zufallszahlenfolge: 2,3,4,5,6,7,8

## Berechnung von Eingabedateien

Für die Veranschaulichung der Berechnung ist folgendes Aktivitätsdiagramm gegeben:



## Erweiterbarkeit

Bisher wurde sich lediglich auf die angegebenen Verfahren bezogen. Für eine Software sollte aber stets der Aspekt der Erweiterbarkeit beachtet werden. Somit sollten folgende Punkte beachtet werden, bevor weiter Verfahren wird:

* Es können weitere Generatoren eingebunden werden
* Es können weitere Gütefunktionen eingebunden werden

Wichtig ist zu beachten, dass für jeden Generator und jede Gütefunktion eigene Bedingungen sowie Parameter vorhanden sind.

## Sonderfälle

Durch die zuvor analysierten Eigenschaften und der Aufgabenbeschreibung ergeben sich folgende Sonderfälle, die beachtet werden müssen:

* Es werden keine Parameter angegeben
  + Die Anwendung gibt eine Fehlermeldung aus
* Bei der Anwendung der Polar-Methode ist kein Generator für die initiale Zufallszahlengenerierung angegeben
  + Die Anwendung nutzt die interne Klassenbibliothek, um zwei initiale Zufallszahlen zu generieren
* Die angegebenen Parameter sorgen für eine zu hohe Durchführungszeit
  + Die Anwendung beschränkt den Wertebereich für die Eingabe von Parameter, um zu lange Ausführungszeiten zu vermeiden

## Fehlerfälle

Beim Lesen und Schreiben von Dateien kann es zu technischen, syntaktischen und semantischen Fehlern führen. Diese werden im Folgenden erläutert. Detaillierte Informationen aller Fehlerszenarien werden im Kapitel Testdokumentation aufgegriffen.

### Technische Fehler

Technische Fehler entstehen beim direkten Lesen und Schreiben von Dateien, wenn diese zum Beispiel nicht erzeugt werden können oder durch fehlende Rechte kein Zugriff möglich ist. Geschieht dies beim Lesen, wird für die aktuell zu lesende Datei eine passende Fehlerinformation in eine Ausgabedatei erzeugt. Beim Schreiben wird eine dazugehörige Information auf der Konsolenausgabe inklusive Inhalt der Datei angegeben.

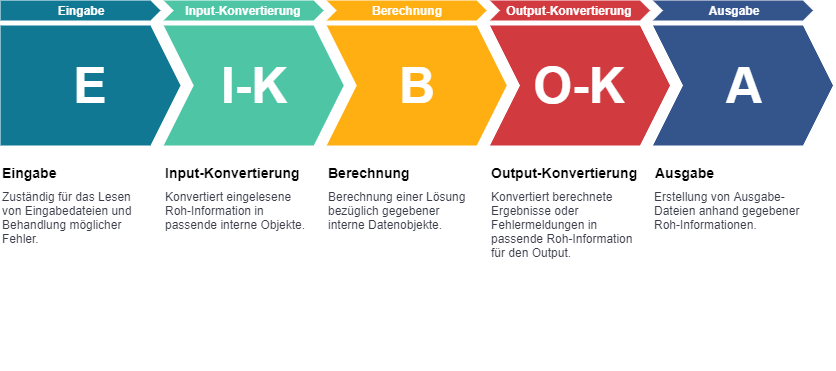
### Syntax- und Semantikfehler

Syntax- und Semantikfehler treten nach dem direkten Lesen der Datei auf, sobald die Rohinformation für die Erzeugung von internen Objekten genutzt wird. Sollten bei diesem Vorgang Fehler auftreten, wird eine passende Fehlermeldung in eine Ausgabedatei generiert.

Mögliche Ursachen solcher Fehler können unbekannte Wörter, Zeichen oder Ausdrücke sein, die nicht extrahiert werden können.

# Verfahrenbeschreibung

Durch die Analyse ergibt sich folgende Aufgabeneinteilung des Presenters ergibt sich durch folgende prozessualen Ablauf:



* **Eingabe**

Die Eingabe dient zum Lesen von Eingabedateien und Behandlung möglicher Fehler.

* **Input-Konvertierung**

Dies dient zur Extrahierung von wichtigen Informationen aus der gesammelten Roh-Information nach dem Lesen der Eingabe-Datei.

* **Berechnung**

Die Berechnungsinstanz berechnet abhängig von den konvertierten Daten gesuchte Ergebnisse, wie zum Beispiel eine Erzeugung von Zufallszahlen mittels eines Generators oder der Bewertung von Zufallszahlen mittels einer Gütefunktion.

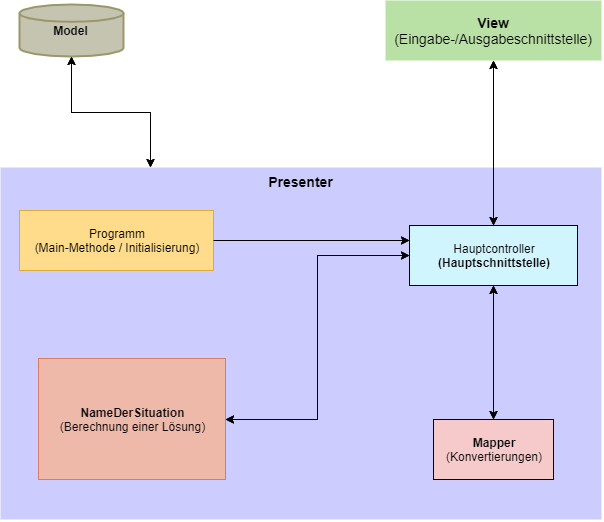
* **Output-Konvertierung**

Nachdem ein Ergebnis ermittelt wurde, wird dies in eine passende Roh-Information für die Ausgabe konvertiert.

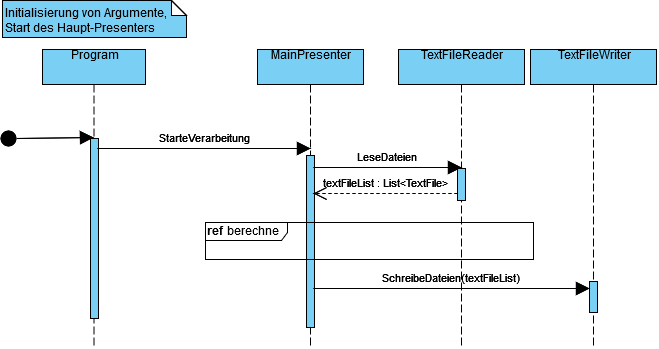
* **Ausgabe**

Die Ausgabe dient abschließend zur Erzeugung von Ausgabe-Dateien mit der dazugehörigen Roh-Information.

Für eine Logische Trennung der Daten und Aufgaben wird die Anwendung mittels eines Model-View-Presenter-Pattern entwickelt. So kann die Verarbeitung zugunsten der Kohäsion über eine – oder auch mehrere – Presenter-Klassen aufgeteilt werden, um Teilaufgaben getrennt zu behandeln. Verarbeitende Daten werden mittels Model-Klassen gesichert bzw. Verwaltet. Die View-Klassen behandeln die Eingabe und Ausgabe der Informationen für den Benutzer. Dies lässt sich im groben wie folgt darstellen:



Mithilfe des oben beschriebenen Systems kann folglich ein Überblick für den High-Level-Datenfluss erzeugt werden:



* Programm-Klasse verwaltet übergebene Argumente und startet die Haupt-Verarbeitung über die MainPresenter-Klasse
* Die MainPresenter-Klasse liest über den TextFileReader die Dateien in einem Pfad ein und erhält eine Liste aller Dateien
* Im Anschluss wird die Berechnung durchgeführt
* Zuletzt werden die Ergebnisse über die TextFileWriter-Klasse in Dateien gesichert

## Logische Datenstruktur

Folgend werden die wichtigsten Klassen aufgelistet, um ein gesamter Überblick der Anwendung zu erhalten.

### Program

* Haupteinstiegspunkt des Programms
* Verwaltet Initialisierung relevanter Argumente
* Startet den Hauptverarbeitungspresenter für den Zufallgenerator

### MainPresenter

* Hauptverarbeitungspresenter als Schnittstelle zwischen Daten lesen, verarbeiten und schreiben.
* Oberste Schicht für die Verwaltung von Ausnahmen

### ZufallData

* Datenobjekt für die Datenhaltung eingelesener Roh-Information, welche für den Zufallsgenerator relevant sind.
  + Berechnungsziel, Generatortyp, Bewertungstyp, Parameterliste, Zufallzahlen

### ZufallErgebnisData

* Datenobjekt für die Datenhaltung ausgerechneter Ergebnisse
  + Zufallszahlen, Bewertung

### TextFile

* Datenobjekt für die Datenhaltung eingelesener oder auszugebener Roh-Informationen
  + Name, Inhalt

### TextFileReader

* Klasse zum Lesen von Dateien eines Eingabepfads
* Gibt eine Liste aller Dateiinformationen in Form von TextFile-Objekten zurück

### TextFileWriter

* Schreibt Ausgabeinformationen in Form von TextFile-Objekten in einen Ausgabepfad

### ZufallDataInputMapper

* Mapper für die Konvertierung eingelesener Roh-Information in ein ZufallData-Objekt
  + Syntaxfehler werden hier ersichtlich und behandelt

### ZufallErgebnisOutputMapper

* Mapper für die Konvertierung berechneter Ergebnisse in Form von ZufallErgebnisData-Objekten in Roh-Informationen für die Ausgabe

### ZufallsgeneratorPresenter

* Presenter für die Zufalldatenerzeugung
* Benötigt beim Initialisieren ein ZufallData-Objekt, um Ergebnisse in einem ZufallErgebnisData-Objekt zu sichern

### CalculationException

* Ausnahme-Klasse für Berechnungsfehler
  + Wird geworfen, wenn eine Berechnung aufgrund eines Parameters oder anderer Probleme nicht erfolgen kann

### ZufallMappingException

* Oberste Ausnahme-Klasse für Mapping-Fehler
  + Wird geworfen, wenn das Konvertieren fehlschlägt

### ParameterException

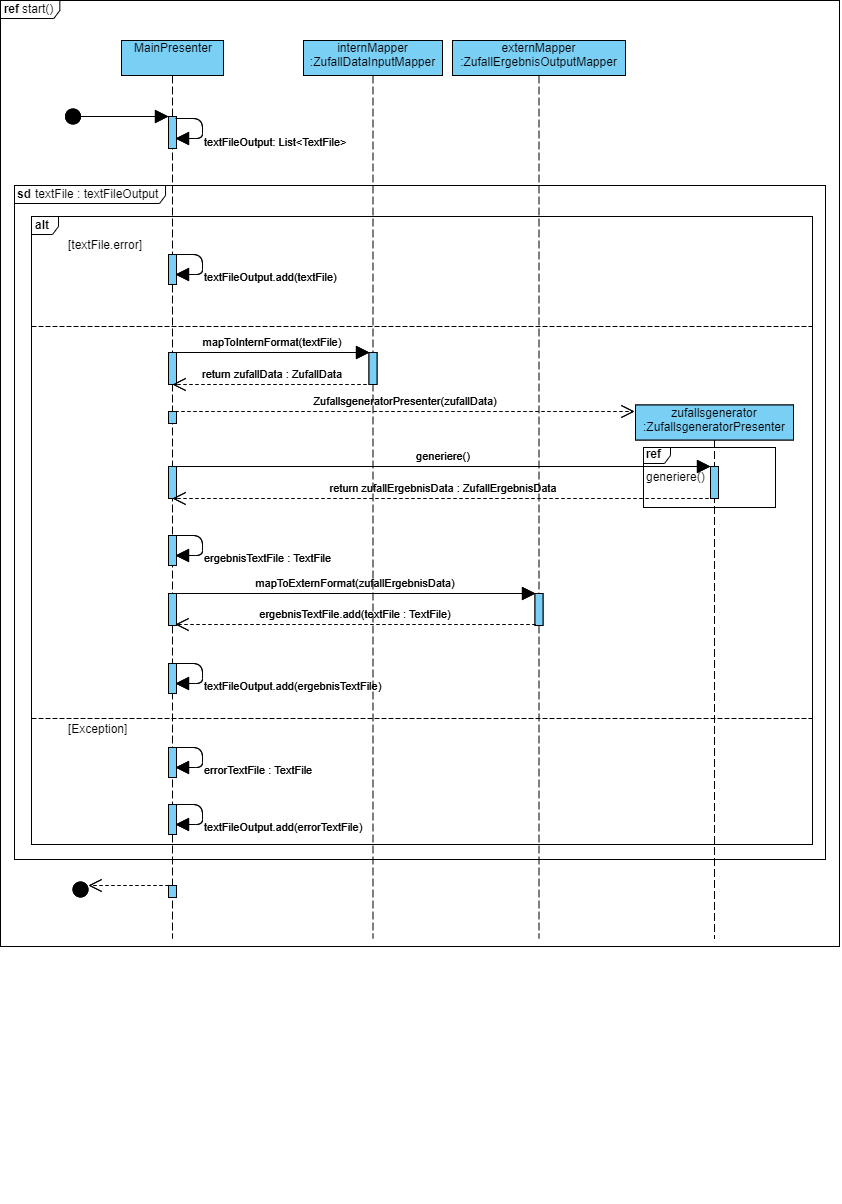
* Erbt von ZufallMappingException und ist somit eine detailliertere Ausnahme
  + Wird geworfen, wenn spezielle Stellen in einer Textdatei nicht gelesen werden können
  + Wird auf oberster Schicht von ZufallMappingException behandelt (weitergegeben)

## Systembeschreibung

Dieses Kapitel behandelt die Veranschaulichung des Systems der Anwendung in Form von Sequenzdiagrammen, UML-Klassendiagrammen und Nassi-Shneiderman-Diagrammen, um einen detaillierten Überblick der Anwendung zu erhalten.

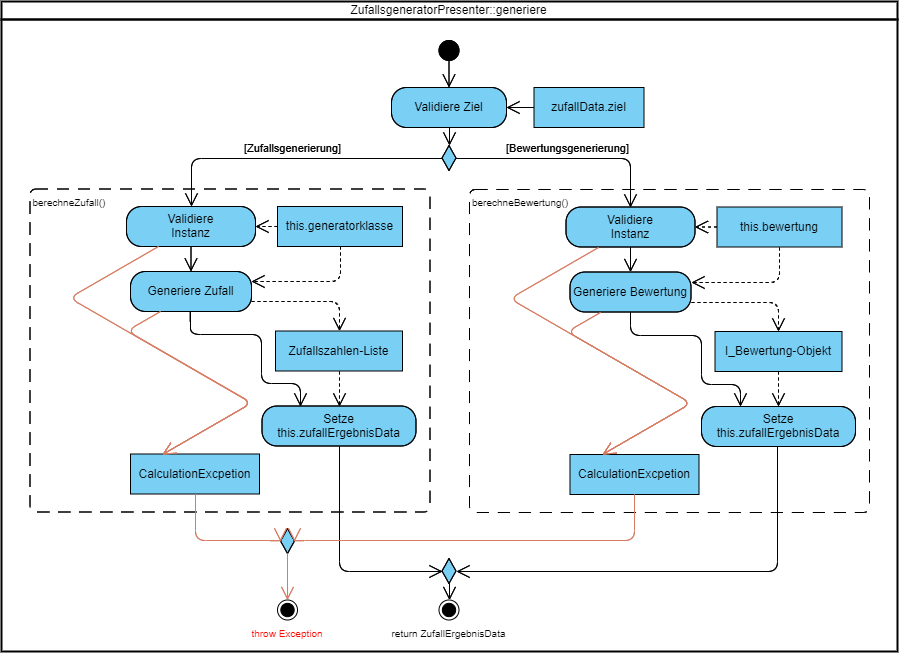
### Allgemeiner Datenfluss

Folgende Abbildung beschreibt den technischen Datenfluss hinsichtlich der Eingabe, Konvertierungen und Ausgabe von Informationen:



Mittels einer textFileOutput-Liste werden die Ergebnisse eingelesener Daten iterativ hinzugefügt. Sobald beim Verarbeitungsprozess ein Fehler entsteht, wird ein alternatives TextFile-Objekt der textFileOutput-Liste hinzugefügt.

Folgend ist die Verarbeitung des Zufallsgenerators dargestellt:



Diese unterscheidet bei der Verarbeitung zwischen zwei Use-Cases:

* Generiere Zufallszahlen mittels der berechneZufall-Methode
* Generiere Bewertungen mittels der berechneBewertung-Methode

Wie bereits beschrieben, können beim Berechnen CalculationException-Objekte geworfen werden, wenn Fehler vorhanden sind.

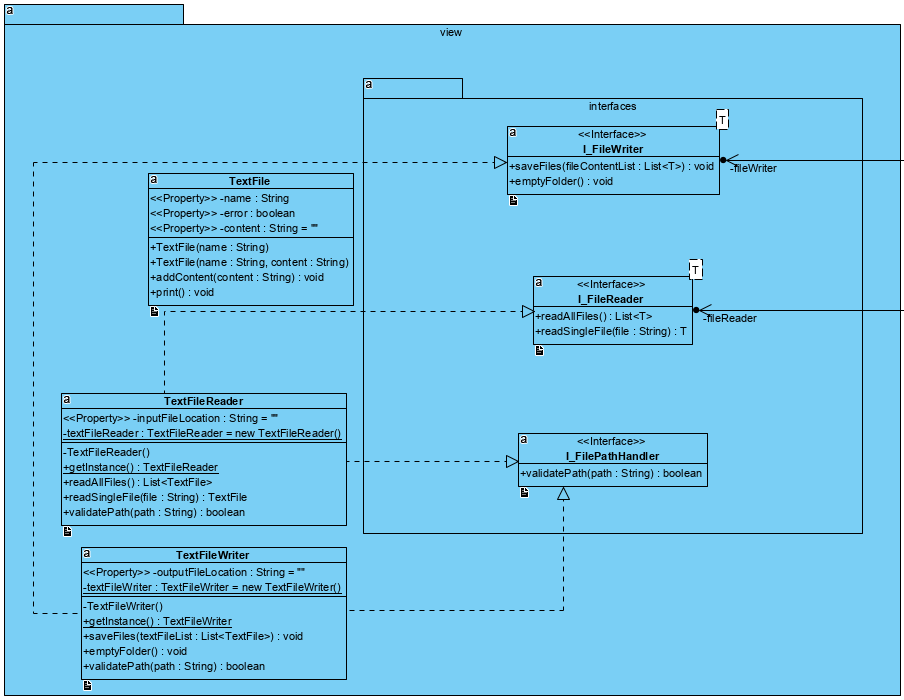
### ULM-Klassendiagramm

Folgende Abbildungen zeigen jeweils Teile aus dem UML-Klassendiagramm. Es wird zudem jeweils auf die Abweichungen eingegangen.

Das gesamte UML-Diagramm kann mittel Visual Paradigm eingesehen werden. Dazu die Datei unter <data \UML.vpp> öffnen.

#### View

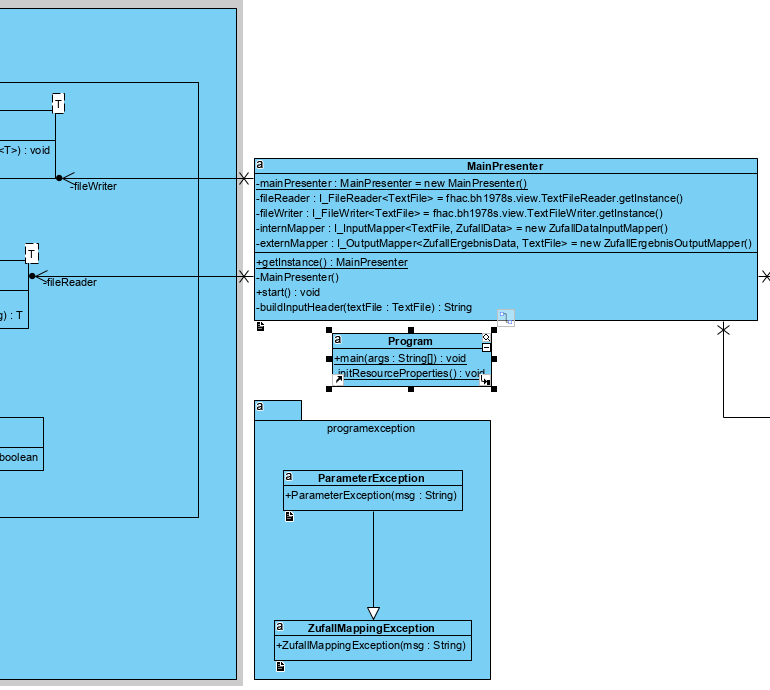
Die View beinhaltet Klassen für die Verwaltung von Ein- und Ausgabedateien:



Zu sehen sind für die Eingabe und der Ausgabe Interfaces, welche in Form von TextFileReader und TextFileWriter implementiert werden. Diese verwenden TextFile-Objekte, welche für das Lesen und Schreiben von Textdateien zuständig sind. Vorteil ist, dass die Anwendung um weitere Implementierungen erweitert werden kann, falls ein alternatives Dateiformat für den Prozess genutzt werden soll.

#### Program und Hauptpresenter

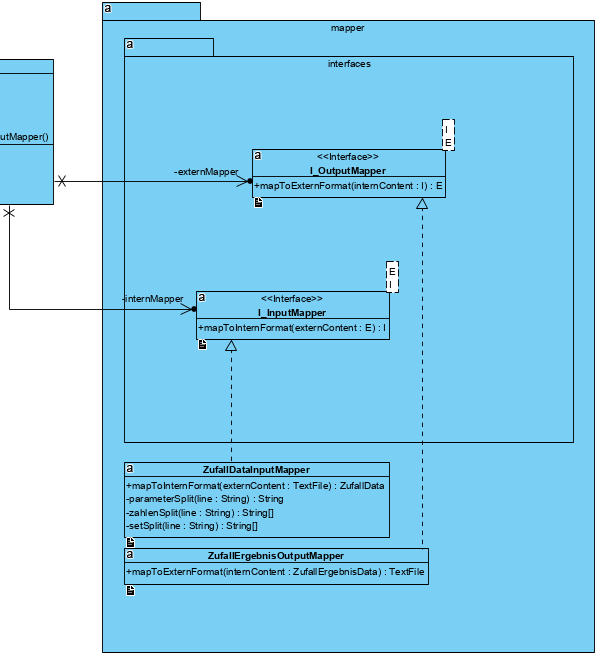
Folgend ein Ausschnitt für die Hauptverarbeitung innerhalb der Presenter-Klasse sowie dem Einsteigspunkt in der Program-Klasse sowie Exceptions:



Zu sehen ist die MainPresenter-Klasse, welche – wie zuvor beschrieben – die Schnittstelle zur Eingabe, Ausgabe und Verwaltung von Dateien zuständig ist. Diese implementiert die vorher gezeigten Interfaces. Des Weiteren sind die Exception-Klassen zu sehen, welche beim Konvertieren geworfen werden können.

#### Mapper-Presenter

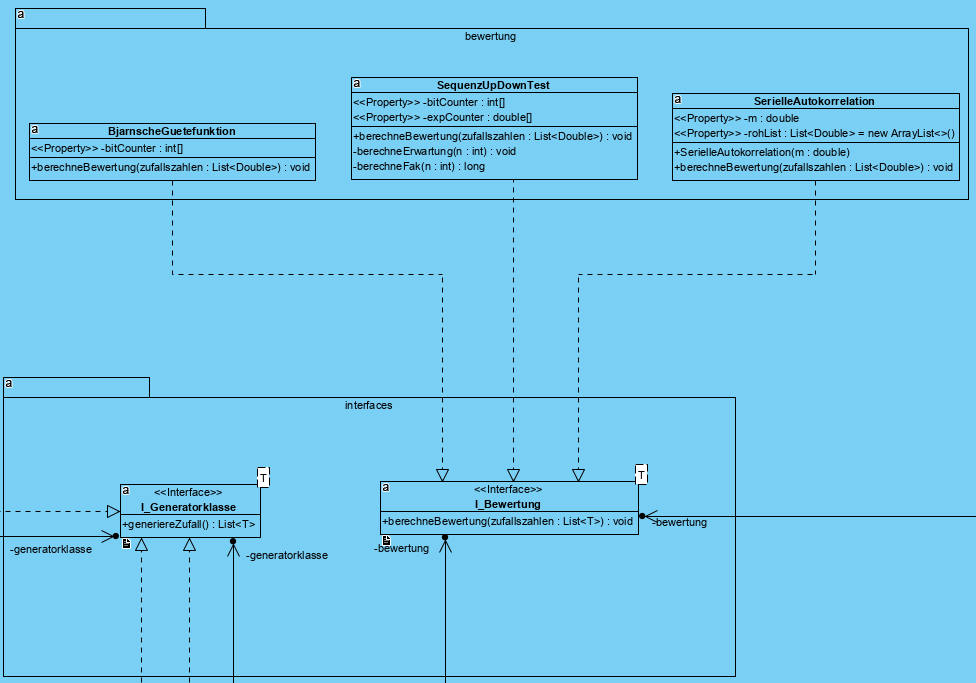
Folgend sind die Mapper-Klassen beschrieben:



Wie bei der Ein- und Ausgabe sind diese mittels Interfaces implementiert, sodass in Abhängigkeit von den Eingabedateien diese passend konvertiert werden können.

#### Bewertung-Presenter

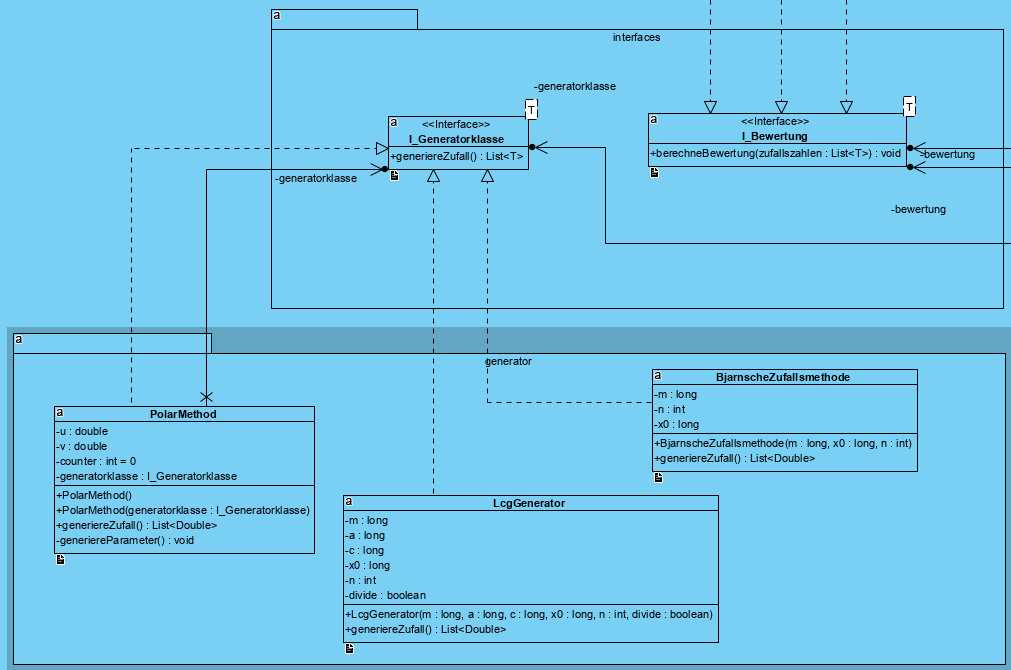
Der Zufallsgenerator besitzt für die Bewertungen ein Interface, welches von jeder – auch in Zukunft zu implementierenden Bewertungen – implementiert wird:



Jede Bewertung besitzt vom Interface eine Haupt-Berechnungsmethode, welche durch jeweilige implementierende Klasse durch weitere für die Berechnung wichtige Methoden ergänzt wird.

#### Generator-Presenter

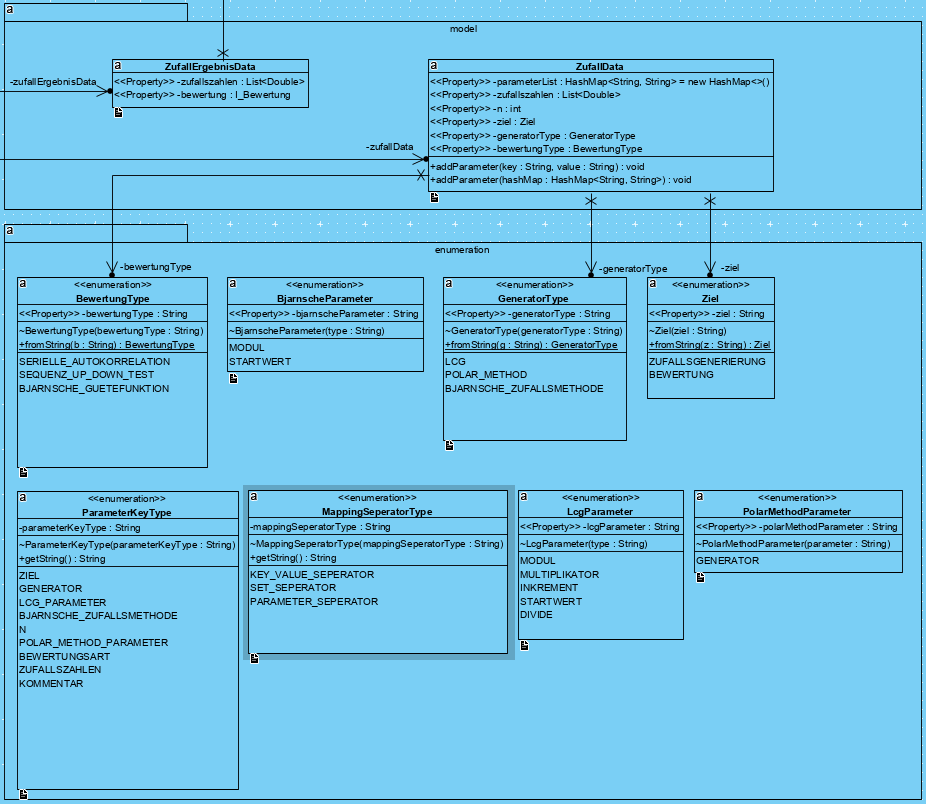
Ähnlich wie der Bewertung-Presenter wurde beim Generator-Presenter ein Interface genutzt, welches von den jeweiligen Generatoren implementiert wird:



Das Interface besitzt eine zentrale Berechnungsmethode, und wird auch durch Generatorspezifischen Eigenschaften ergänzt.

#### Model

Im Folgenden sind die Zufallsgenerator-Model-Klassen inklusive Enumeration zu sehen:

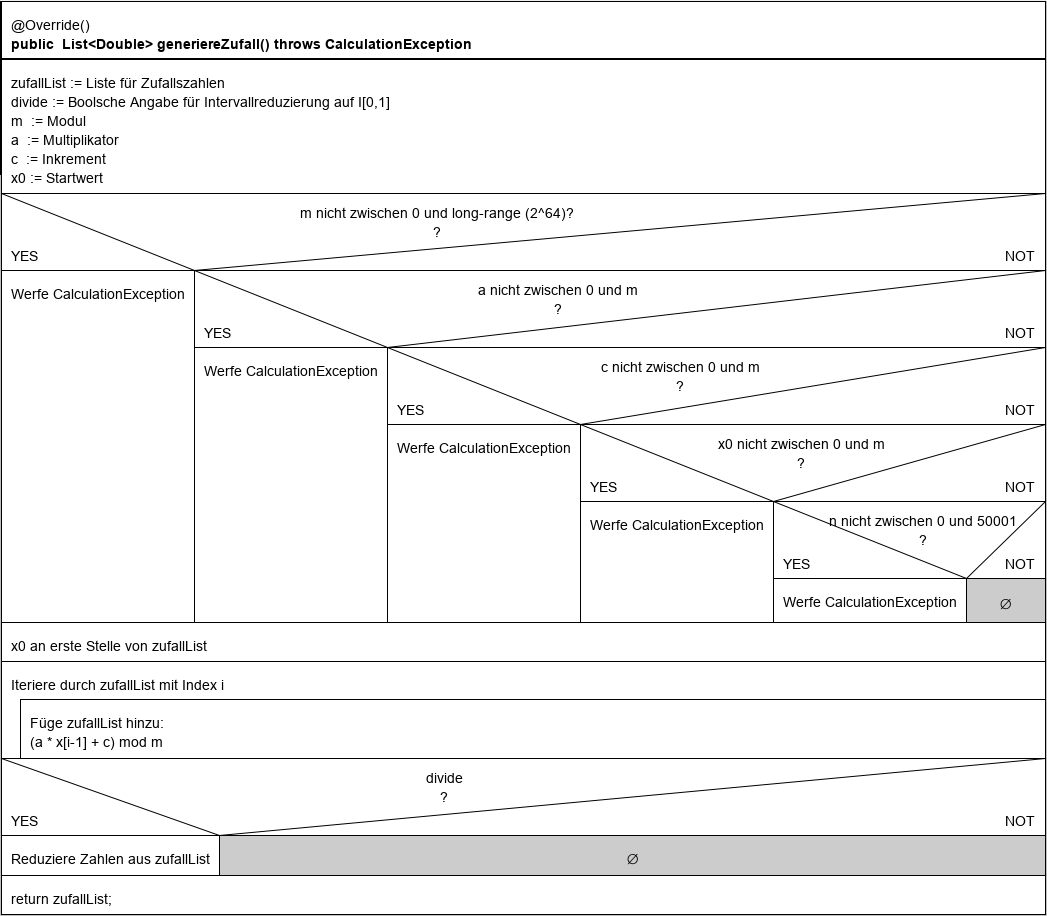


Die Enumeration wurden im ersten Konzept nicht modelliert aber im Nachhinein aufgrund der String-Komplexität beim Lesen und Verarbeiten entwickelt, um eine bessere Übersicht und Verwaltung des Datenflusses zu gewährleisten.

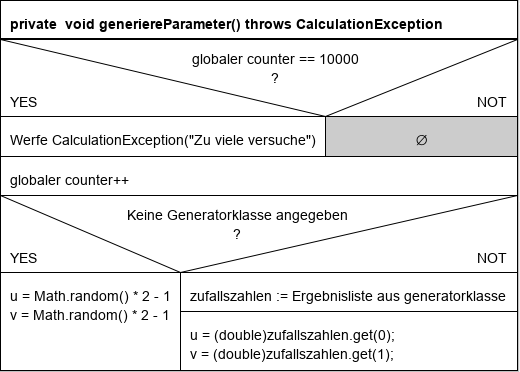
### Methoden-Datenfluss

Der Datenfluss einzelner Methoden wird in Form von Nassi-Shneiderman-Diagrammen angegeben.

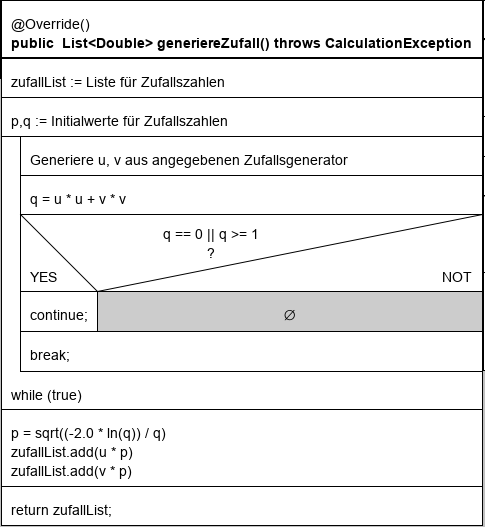
#### LcgGenerator::generiereZufall



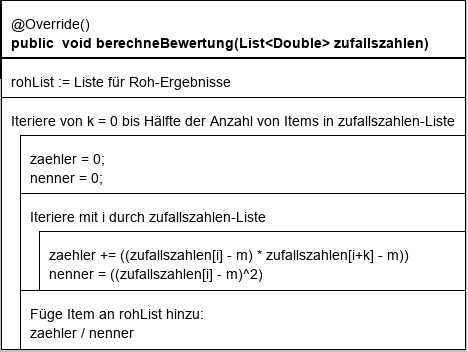
#### PolarMethode::GeneriereParameter



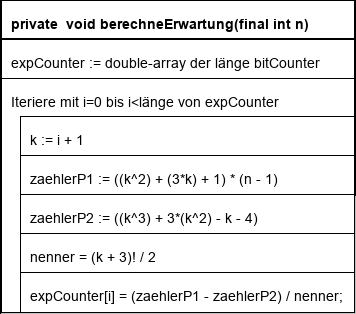
#### LcgGenerator::generiereZufall



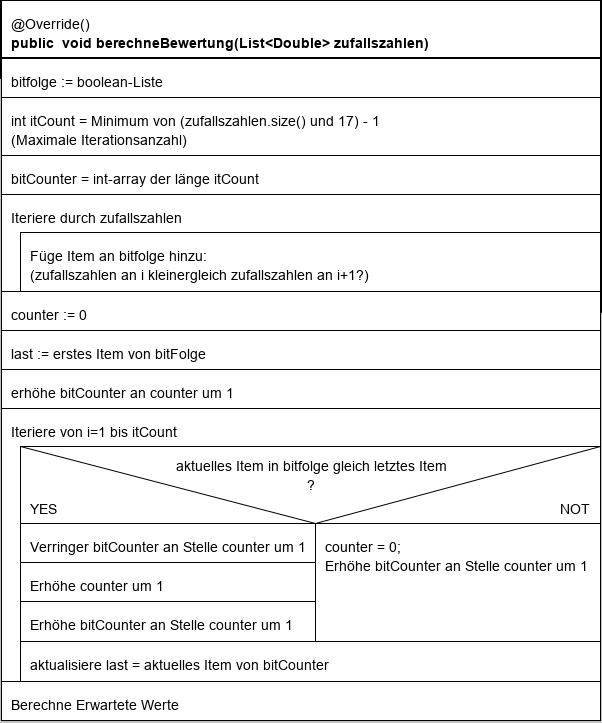
#### SerielleAutokorrelation::berechneBewertung



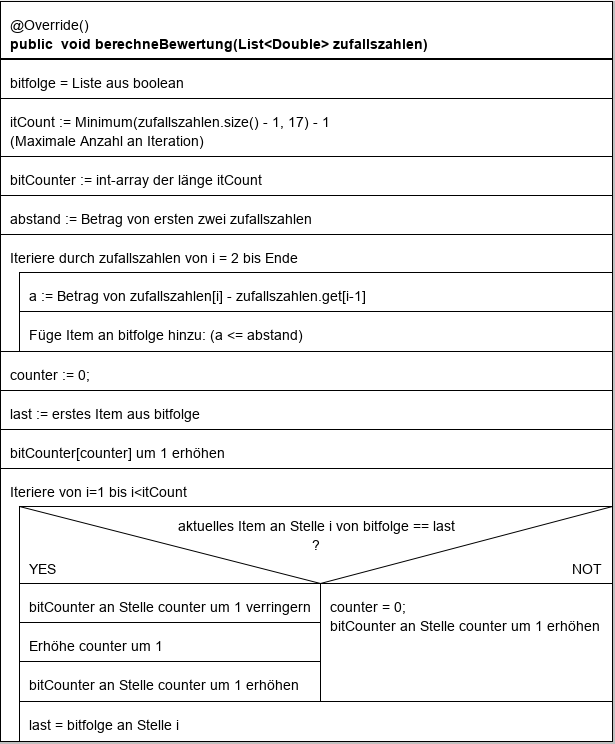
#### SequenzUpDownTest::berechneErwartung



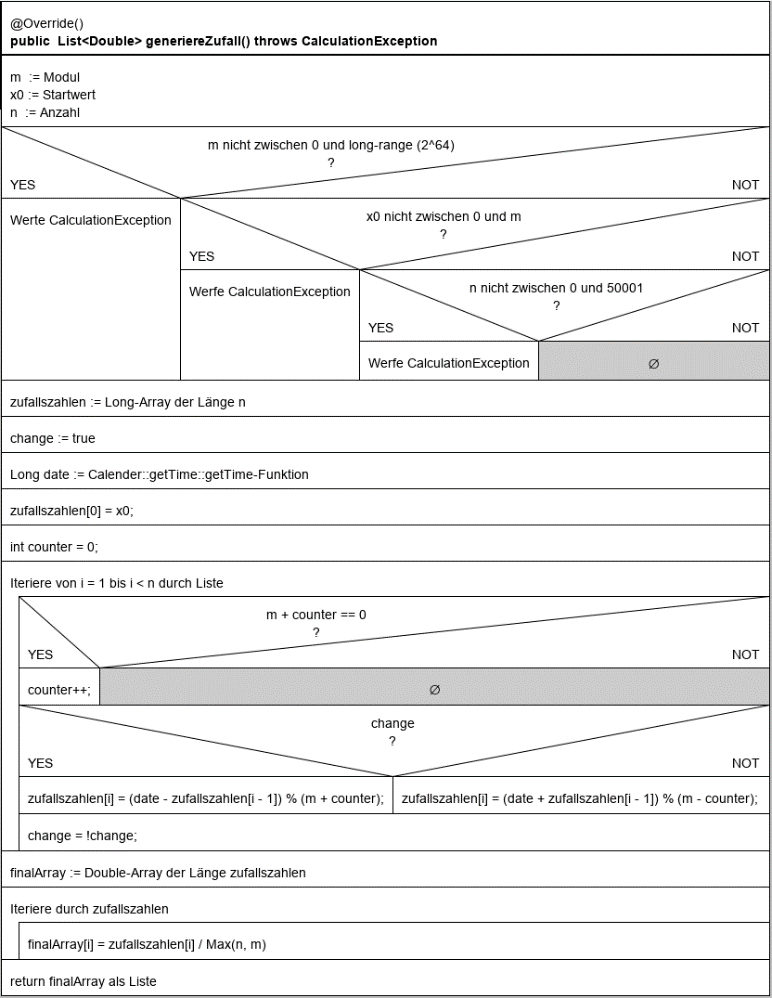
#### SequenzUpDownTest::berechneBewertung



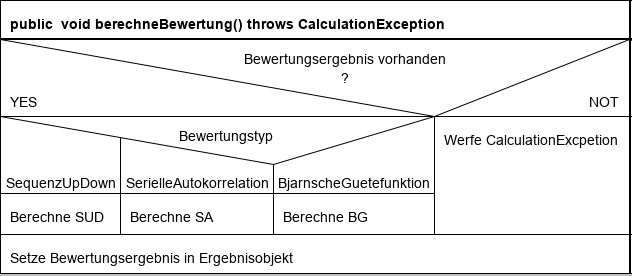
#### BjarnscheGuetefunktion::berechneBewertung



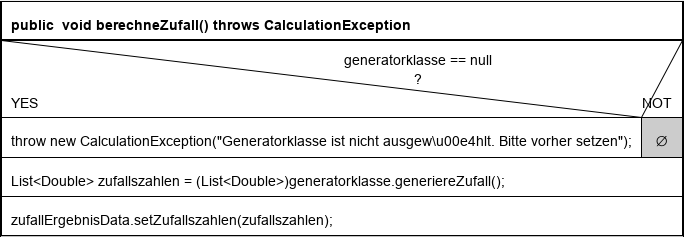
#### BjarnscheZufallsmethode::generiereZufall



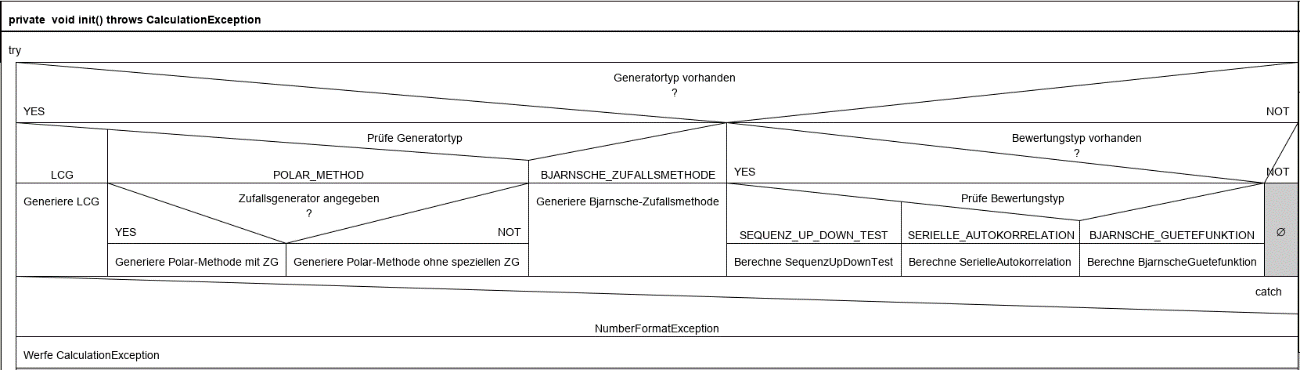
#### ZufallsgeneratorPresenter::berechneBewertung



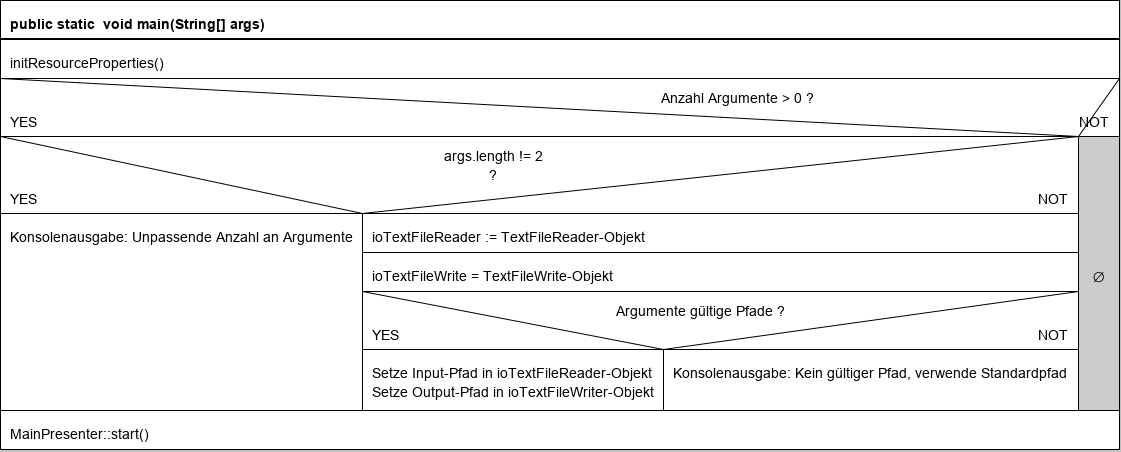
#### ZufallsgeneratorPresenter::berechneBewertung



#### ZufallsgeneratorPresenter::init



#### Program::main



# Use-Cases

Folgend werden die einzelnen Use-Cases mit dazugehörigen Eingaben vorgestellt.

## Use-Case 1: Zufallsgenerierung

Dieser Use-Case beschäftigt sich mit der Zufallsgenerierung mithilfe angegebener Zufallsgeneratoren. Für jeden Generator existieren spezifische Parameter für die Berechnung. Generell gilt als Einschränkung als maximale Zahlengenerierung n=50000 sowie generelle Werte von maximal 264.

### LCG-Generator

Für den LCG-Generator ist zu beachten, dass bei der Ausgabe beliebig in Abhängigkeit von dem Parameter n viele Zahlen – also Zeilen – generiert werden.

*Eingabe:*

# Verwendung des LCG-Generators

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=<Ganzzahl>,a=<Ganzzahl>,c=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>, divide=<Wahrheitsaussage>

*Ausgabe:*

-----INPUT-----

# Verwendung des LCG-Generators

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=<Ganzzahl>,a=<Ganzzahl>,c=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>, divide=<Wahrheitsaussage>

-----OUTPUT-----

Zufallszahlen:

<Index> : <Zahl>

….

### Polar-Methode

Für die Polar-Methode ergeben sich immer zwei Zufallszahlen. Die Polar-Method-Parameter-Angabe ist optional. Wird diese nicht mitgegeben, so wird intern mithilfe der Java-Bibliothek initiale Zufallszahlen generiert. Je nach mitgegebenen Polar-Method-Parameter können sich weitere Eingaben für diesen Parameter ergeben.

*Eingabe:*

# Verwendung der Polar-Methode

Ziel:Zufallsgenerierung

Generator:Polar-Methode

Polar-Method-Parameter:Generator=Linear-Kongruenz-Generator

LCG-Parameter:m=<Ganzzahl>,a=<Ganzzahl>,c=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>, divide=<Wahrheitsaussage>

*Ausgabe:*

-----INPUT-----

# Verwendung der Polar-Methode

Ziel:Zufallsgenerierung

Generator:Polar-Methode

Polar-Method-Parameter:Generator=Linear-Kongruenz-Generator

LCG-Parameter:m=<Ganzzahl>,a=<Ganzzahl>,c=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>, divide=<Wahrheitsaussage>

-----OUTPUT-----

Zufallszahlen:

0 : <Zahl>

1 : <Zahl>

### Bjarnsche-Zufallsmethode

Ähnlich wie beim LCG-Generator müssen hier diverse Parameter angegeben. Die Ausgabe besitzt auch in Abhängigkeit von n beliebig viele Zeilen.

*Eingabe:*

# Verwendung der Bjarnsche-Zufallsmethode

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>

*Ausgabe:*

-----INPUT-----

# Verwendung der Bjarnsche-Zufallsmethode

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>

-----OUTPUT-----

Zufallszahlen:

<Index> : <Zahl>

….

## Use-Case 2: Bewertung

Dieser Use-Case beschäftigt sich mit der Bewertung mithilfe angegebener Bewertungsart sowie Zufallszahlen.

### Sequenz-Up-Down-Test

Für den Sequenz-Up-Down-Test ergeben sich bis maximal k=17, da es sonst zu einem überlauf bei der Fakultätsberechnung kommt.

*Eingabe:*

# Verwendung der Bjarnsche-Zufallsmethode

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=<Ganzzahl>,x0=<Ganzzahl>,n=<Ganzzahl>

*Ausgabe:*

-----INPUT-----

# Bewertung SequenzUpDownTest

Ziel:Bewertung

Bewertungsart:Sequenz-Up-Down-Test

Zufallszahlen:Ausgabe:<Zahl>, …

-----OUTPUT-----

Sequenz-Up-Down-Test-Ergebnis:

k Calc: N(k) Exp: N(k)

<Index> <Ganzzahl> <Fließkommazahl>

….

### Serielle-Autokorrelation

Die Serielle-Autokorrelation iteriert bis zur Hälfte der angegebenen Zufallszahlen. Grund dafür ist die stochastische Notwendigkeit, da für höhere Zahlenabstände keine sinnvolle Aussage getroffen werden kann.

*Eingabe:*

# Bewertung mit Serielle-Autokorrelation

Ziel:Bewertung

Bewertungsart:Serielle-Autokorrelation

Zufallszahlen:<Zahl>,…

*Ausgabe:*

-----INPUT-----

# Bewertung mit Serielle-Autokorrelation

Ziel:Bewertung

Bewertungsart:Serielle-Autokorrelation

Zufallszahlen:<Zahl>,…

-----OUTPUT-----

Serielle-Autokorrelation-Ergebnis:

k Calc: N(k)

<Index> <Ganzzahl>

….

### Bjarnsche-Gütefunktion

Für die Bjarnsche-Gütefunktion ergeben sich ebenfalls maximal k=17 Iterationen.

*Eingabe:*

# Bewertung mit Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen:<Zahl>,…

*Ausgabe:*

-----INPUT-----

# Bewertung mit Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen:<Zahl>,…

-----OUTPUT-----

Bjarnsche-Guetefunktion-Ergebnis:

k Calc: N(k)

<Index> <Ganzzahl>

….

# Testdokumentation

Für die Testdokumentation ergeben sich 43 Tests. Diese werden in Positivtests und Negativtests unterteilt.

Es werden jeweils nur die Ausgabedateien aufgelistet, da diese die Information der Eingabe beinhalten.

## Positivtests

Positivtests beschreiben Fälle, in denen eine Berechnung erfolgreich ist.

### Test01\_LcgBerechnung\_SimpelNoDivide

Testet den LCG-Generator mit einfachen Parametern.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit einfachen Parametern

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=10,a=2,c=0,x0=1,n=10, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 1.0

1 : 2.0

2 : 4.0

3 : 8.0

4 : 6.0

5 : 2.0

6 : 4.0

7 : 8.0

8 : 6.0

9 : 2.0

### Test02\_LcgBerechnung\_SimpelDivide

Testet den LCG-Generator mit einfachen Parametern inklusive positiven Teilungsparameter, um das Ergebnis auf das Intervall I[0;1] einzuschränken.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit einfachen Parametern und Einschraenkung auf I[0;1]

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=10,a=2,c=0,x0=1,n=10, divide=true

-----OUTPUT-----

Zufallszahlen:

0 : 0.1

1 : 0.2

2 : 0.4

3 : 0.8

4 : 0.6

5 : 0.2

6 : 0.4

7 : 0.8

8 : 0.6

9 : 0.2

### Test03\_Lcg\_ANSIC

Testet den LCG-Generator mit ANSI-C Parametern.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit ANSI-C Parameter

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483648,a=1103515245,c=12345,x0=12345,n=17, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 12345.0

1 : 1.406932606E9

2 : 6.54583775E8

3 : 1.449466924E9

4 : 2.29283573E8

5 : 1.109335178E9

6 : 1.051550459E9

7 : 1.293799192E9

8 : 7.94471793E8

9 : 5.5118831E8

10 : 8.03550167E8

11 : 1.772930244E9

12 : 3.70913197E8

13 : 6.39546082E8

14 : 1.381971571E9

15 : 1.695770928E9

16 : 2.121308585E9

### Test04\_Lcg\_ANSIC\_Divide

Testet den LCG-Generator mit ANSI-C Parametern sowie positiven Teilungsfaktor für Intervalleinschränkung auf I[0;1].

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit ANSI-C Parameter und Einschraenkung auf I[0;1]

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483648,a=1103515245,c=12345,x0=12345,n=17, divide=true

-----OUTPUT-----

Zufallszahlen:

0 : 5.748588591814041E-6

1 : 0.6551540484651923

2 : 0.30481432331725955

3 : 0.6749606337398291

4 : 0.10676848376169801

5 : 0.5165744470432401

6 : 0.4896663404069841

7 : 0.6024721972644329

8 : 0.36995475785806775

9 : 0.25666705798357725

10 : 0.37418220518156886

11 : 0.8255849797278643

12 : 0.1727199167944491

13 : 0.29781185183674097

14 : 0.6435306607745588

15 : 0.7896548733115196

16 : 0.9878112864680588

### Test05\_Lcg\_MinimalStandard

Testet den LCG-Generator mit Minimal-Standard-Parameter.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit MinimalStandard-Parameter

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483647,a=16807,c=0,x0=1,n=1000, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 1.0

1 : 16807.0

2 : 2.82475249E8

3 : 1.622650073E9

4 : 9.84943658E8

5 : 1.14410893E9

6 : 4.70211272E8

7 : 1.01027544E8

8 : 1.457850878E9

9 : 1.458777923E9

10 : 2.007237709E9

11 : 8.2356444E8

12 : 1.115438165E9

13 : 1.784484492E9

14 : 7.4243042E7

….

### Test06\_Lcg\_RANDU

Testet den LCG-Generator mit RANDU-Parameter.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit RANDU-Parameter

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483648,a=65539,c=0,x0=1,n=400, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 1.0

1 : 65539.0

2 : 393225.0

3 : 1769499.0

4 : 7077969.0

5 : 2.6542323E7

6 : 9.5552217E7

7 : 3.34432395E8

8 : 1.146624417E9

9 : 1.722371299E9

10 : 1.4608041E7

11 : 1.766175739E9

12 : 1.875647473E9

13 : 1.800754131E9

14 : 3.66148473E8

### Test07\_Lcg\_SIMSCRIPT

Testet den LCG-Generator mit SIMSCRIPT-Parameter.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit NAG's LCG-Parameter

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=576460752303423488,a=302875106592253,c=0,x0=12345678,n=100, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 1.2345678E7

1 : 2.74100763628089376E17

2 : 1.04745623391925184E17

3 : 1.7984134677309664E17

4 : 3.828183385741296E17

5 : 2.9957680316340813E17

6 : 2.48549203763179296E17

7 : 3.311108359842372E16

8 : 2.21005936772050944E17

9 : 7.1948582498730968E16

10 : 1.37132219212724352E17

11 : 4.3498929433798515E17

12 : 2.68922530559853664E17

13 : 2.71200146465457856E17

14 : 3.003257982864711E17

### Test08\_Lcg\_NAGsLCG

Testet den LCG-Generator mit NAG’s-LCG-Parameter.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit SIMSCRIPT-Parameter

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483647,a=630360016,c=0,x0=1,n=821, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 1.0

1 : 6.30360016E8

2 : 1.54903533E9

3 : 2.64620982E8

4 : 5.29512731E8

5 : 1.896697821E9

6 : 2.116530888E9

7 : 1.923129168E9

8 : 1.674201058E9

9 : 1.08088067E8

10 : 8.59154222E8

11 : 1.946499387E9

12 : 1.377890442E9

13 : 1.38279331E9

14 : 7.68302678E8

### Test09\_Lcg\_MaplesLCG

Testet den LCG-Generator mit Maple’s-LCG-Parameter.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit Maple's LCG-Parameter

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=999999999989,a=427419669081,c=0,x0=1,n=250, divide=false

-----OUTPUT-----

Zufallszahlen:

0 : 1.0

1 : 4.27419669081E11

2 : 3.2111069327E11

3 : 3.43633073697E11

4 : 4.74256143563E11

5 : 5.58458718976E11

6 : 7.46753830538E11

7 : 3.2062222085E10

8 : 7.22974121768E11

9 : 6.04305613921E11

10 : 7.45580037409E11

11 : 2.59811952655E11

12 : 3.10075487163E11

13 : 7.97179490457E11

14 : 3.916959416E10

### Test11\_PolarMethodeDefault

Testet die Polar-Methode ohne Generatorübergabe.

*Ausgabe:*

-----INPUT-----

# Teste Polar Methode ohne Generatorangabe (Verwendung von Math.random)

Ziel:Zufallsgenerierung

Generator:Polar-Methode

-----OUTPUT-----

Zufallszahlen:

0 : 0.9090262047693153

1 : -1.1545139929596084

### Test12\_PolarMethode\_Lcg\_ANSIC

Testet die Polar-Methode mit LCG als Generatorübergabe.

*Ausgabe:*

-----INPUT-----

# Teste Polar Methode mit ANSI-C LCG-Verfahren

Ziel:Zufallsgenerierung

Generator:Polar-Methode

Polar-Method-Parameter:Generator=Linear-Kongruenz-Generator

LCG-Parameter:m=2147483648,a=1103515245,c=12345,x0=12345,n=17, divide=true

-----OUTPUT-----

Zufallszahlen:

0 : 1.1411924119971343E-5

1 : 1.3005919920279903

### Test21\_SequenzUpDownTest\_Lcg\_SimpelNoDivide

Testet den Sequenz-Up-Down-Test mit dem Zufallszahlenergebnis aus Test01.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Sequenz-Up-Down-Test mit Ergebnis aus Test01

Ziel:Bewertung

Bewertungsart:Sequenz-Up-Down-Test

Zufallszahlen:1.0,2.0,4.0,8.0,6.0,2.0,4.0,8.0,6.0,2.0

-----OUTPUT-----

Sequenz-Up-Down-Test-Ergebnis:

k Calc: N(k) Exp: N(k)

1 0 3.8333333333333335

2 3 1.4166666666666667

3 1 0.34444444444444444

4 0 0.0623015873015873

5 0 0.00882936507936508

6 0 9.975749559082891E-4

7 0 8.818342151675486E-5

8 0 5.461359628026295E-6

9 0 9.185773074661964E-8

### Test22\_SequenzUpDownTest\_ANSI-C

Testet den Sequenz-Up-Down-Test mit dem Zufallszahlenergebnis aus Test03.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung SequenzUpDownTest-Ansi-C-Ergebnis

Ziel:Bewertung

Bewertungsart:Sequenz-Up-Down-Test

Zufallszahlen:12345.0,1.406932606E9,6.54583775E8,1.449466924E9,2.29283573E8,1.109335178E9,1.051550459E9,1.293799192E9,7.94471793E8,5.5118831E8,8.03550167E8,1.772930244E9,3.70913197E8,6.39546082E8,1.381971571E9,1.695770928E9,2.121308585E9

-----OUTPUT-----

Sequenz-Up-Down-Test-Ergebnis:

k Calc: N(k) Exp: N(k)

1 8 6.75

2 2 2.7

3 0 0.7138888888888889

4 1 0.14285714285714285

5 0 0.023065476190476192

6 0 0.003119488536155203

7 0 3.621031746031746E-4

8 0 3.667628667628668E-5

9 0 3.2776508470952917E-6

10 0 2.6015651015651016E-7

11 0 1.837613444756302E-8

12 0 1.1501334252657004E-9

13 0 6.280233214757024E-11

14 0 2.867686399432431E-12

15 0 9.527716250837598E-14

16 0 5.918857377569517E-16

### Test31\_Serielle-Autokorrelation\_Lcg\_SimpleDivide

Testet den Serielle Autokorrelation mit dem Zufallszahlenergebnis aus Test02.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Serielle-Autokorrelation mit Ergebnis aus Test02

Ziel:Bewertung

Bewertungsart:Serielle-Autokorrelation

Zufallszahlen:0.1,0.2,0.4,0.8,0.6,0.2,0.4,0.8,0.6,0.2

-----OUTPUT-----

Serielle-Autokorrelation-Ergebnis:

k Calc: Roh(k)

0 0.9999999999999999

1 0.21428571428571427

2 -0.6363636363636364

3 -0.2608695652173913

4 0.5555555555555555

### Test32\_Serielle-Autokorrelation\_ANSI-C

Testet den Serielle Autokorrelation mit dem Zufallszahlenergebnis aus Test04.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Serielle-Autokorrelation mit Ergebnis aus Test04

Ziel:Bewertung

Bewertungsart:Serielle-Autokorrelation

Zufallszahlen:5.748588591814041E-6,0.6551540484651923,0.30481432331725955,0.6749606337398291,0.10676848376169801,0.5165744470432401,0.4896663404069841,0.6024721972644329,0.36995475785806775,0.25666705798357725,0.37418220518156886,0.8255849797278643,0.1727199167944491,0.29781185183674097,0.6435306607745588,0.7896548733115196,0.9878112864680588

-----OUTPUT-----

Serielle-Autokorrelation-Ergebnis:

k Calc: Roh(k)

0 1.0

1 -0.10130203611347369

2 0.07261047063578632

3 -0.3398616975584141

4 0.38022172663332143

5 0.16680144497823518

6 -0.18178199643761284

7 -0.5457745824783166

8 0.3787061836536131

### Test41\_BjarnscheZufallsmethode

Testet die Bjarnsche-Zufallsmethode mit einfachen Parametern.

*Ausgabe:*

-----INPUT-----

# Teste Berechnung Bjarnsche-Zufallsmethode

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=20,x0=1,n=100

-----OUTPUT-----

Zufallszahlen:

0 : 0.01

1 : 0.1

2 : 0.01

3 : 0.1

4 : 0.01

5 : 0.1

6 : 0.01

7 : 0.1

8 : 0.01

9 : 0.1

10 : 0.01

11 : 0.1

12 : 0.01

13 : 0.1

14 : 0.01

…

### Test51\_BjarnscheGuetefunktion

Testet die Bjarnsche-Gütefunktion mit einfachen Zufallszahlen.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen: 2, 3, 4, 2, 2, 2, 1, 4, 3, 2

-----OUTPUT-----

Bjarnsche-Guetefunktion-Ergebnis:

k Calc: N(k)

0 2

1 3

2 0

3 0

4 0

5 0

6 0

7 0

## Negativtests

Negativtests beschreiben Fälle, in denen eine Berechnung aufgrund der Mapping-Fehler oder Berechnungsfehlern nicht erfolgreich ist. Für die Negativtests wurden jeweils alle Grenzfälle untersucht.

### Test61\_ERR\_KeinZiel

Testet Fehlerszenario, in dem Kein Berechnungsziel gegeben ist.

*Ausgabe:*

-----INPUT-----

# Teste Fehlermeldung: Kein gueltiges Ziel ausgewaehlt

Ziel:ERROR

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen: 2, 3, 4, 2, 2, 2, 1, 4, 3, 2

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Ziel <ERROR> unbekannt.

### Test62\_ERR\_KeinGenerator

Testet Fehlerszenario, in dem kein Generator angegeben ist.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit einfachen Parametern

Ziel:Zufallsgenerierung

Generator:Ich bin kein Generator

LCG-Parameter:m=10,a=2,c=0,x0=1,n=10, divide=false

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Generator <Ich bin kein Generator> unbekannt.

### Test63\_ERR\_KeineBewertungsart

Testet Fehlerszenario, in dem keine Bewertungsart angegeben ist.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Sequenz-Up-Down-Test mit Ergebnis aus Test01

Ziel:Bewertung

Bewertungsart:lol

Zufallszahlen:1.0,2.0,4.0,8.0,6.0,2.0,4.0,8.0,6.0,2.0

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Bewertung <lol> unbekannt.

### Test64\_ERR\_SeperatorType\_01\_A

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit einfachen Parametern

Ziel-Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=10,a=2,c=0,x0=1,n=10, divide=false

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <:> von <Ziel-Zufallsgenerierung> nicht möglich.

### Test64\_ERR\_SeperatorType\_01\_B

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit einfachen Parametern

Ziel:Zufallsgenerierung

Generator'Linear-Kongruenz-Generator

LCG-Parameter:m=10,a=2,c=0,x0=1,n=10, divide=false

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <:> von <Generator'Linear-Kongruenz-Generator> nicht möglich.

### Test64\_ERR\_SeperatorType\_01\_C

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste LCG-Generator mit einfachen Parametern

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter3m=10,a=2,c=0,x0=1,n=10, divide=false

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <:> von <LCG-Parameter3m=10,a=2,c=0,x0=1,n=10, divide=false> nicht möglich.

### Test64\_ERR\_SeperatorType\_01\_D

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste Polar Methode ohne Generatorangabe (Verwendung von Math.random)

Ziel:Zufallsgenerierung

Generator+Polar-Methode

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <:> von <Generator+Polar-Methode> nicht möglich.

### Test64\_ERR\_SeperatorType\_01\_E

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart#Bjarnsche-Guetefunktion

Zufallszahlen: 2, 3, 4, 2, 2, 2, 1, 4, 3, 2

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <:> von <Bewertungsart#Bjarnsche-Guetefunktion> nicht möglich.

### Test64\_ERR\_SeperatorType\_01\_F

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen> 2, 3, 4, 2, 2, 2, 1, 4, 3, 2

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <:> von <Zufallszahlen> 2, 3, 4, 2, 2, 2, 1, 4, 3, 2> nicht möglich.

### Test64\_ERR\_SeperatorType\_02\_A

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste Berechnung Bjarnsche-Zufallsmethode

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=20x0=1,n=100

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Trennung mit <,> von String <m=20x0=1,n=100> nicht möglich. Anzahl an sets stimmt nicht mit Trennungsanzahl überein.

### Test64\_ERR\_SeperatorType\_02\_B

Testet Fehlerszenario, in dem ein Argument vom Parameter nicht extrahiert werden kann.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen: 2+ 3, 4, 2, 2, 2, 1, 4, 3, 2

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Konvertierung von Zahl <2+ 3> in < 2+ 3, 4, 2, 2, 2, 1, 4, 3, 2> nicht möglich.

### Test65\_ERR\_UnbekannteZeile

Testet Fehlerszenario, in dem eine unbekannte Zeile vorhanden ist.

*Ausgabe:*

-----INPUT-----

# Teste Bewertung Bjarnsche-Guetefunktion

Ziel:Bewertung

Bewertungsart:Bjarnsche-Guetefunktion

Zufallszahlen: 2+ 3, 4, 2, 2, 2, 1, 4, 3, 2

-----OUTPUT-----

Fehlermeldung:

Fehler beim lesen eines Parameters. Bitte Eingabedatei überprüfen.

Details:

Konvertierung von Zahl <2+ 3> in < 2+ 3, 4, 2, 2, 2, 1, 4, 3, 2> nicht möglich.

### Test66\_ERR\_LcgBerechnung\_01\_A

Testet Fehlerszenario für LCG-Generator, in dem n zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, n zu gross

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483647,a=630360016,c=0,x0=1,n=50001, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter n befindet sich nicht zwischen 0 und 50000 (inklusiv).

### Test66\_ERR\_LcgBerechnung\_01\_B

Testet Fehlerszenario für LCG-Generator, in dem n zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, n zu klein

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483647,a=630360016,c=0,x0=1,n=0, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter n befindet sich nicht zwischen 0 und 50000 (inklusiv).

### Test66\_ERR\_LcgBerechnung\_02\_A

Testet Fehlerszenario für LCG-Generator, in dem m zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, m zu gross

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=18446744073709551616,a=630360016,c=0,x0=1,n=100, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

Konvertierung von LCG-Parameter mit Wert <18446744073709551616> nicht möglich.

### Test66\_ERR\_LcgBerechnung\_02\_B

Testet Fehlerszenario für LCG-Generator, in dem m zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, m zu klein

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=0,a=630360016,c=0,x0=1,n=677, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter m befindet sich nicht zwischen 0 und 2^64 (long-range)

### Test66\_ERR\_LcgBerechnung\_03\_A

Testet Fehlerszenario für LCG-Generator, in dem a zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, a zu gross

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=10,a=10,c=0,x0=1,n=60, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter a befindet sich nicht zwischen 0 und m.

### Test66\_ERR\_LcgBerechnung\_03\_B

Testet Fehlerszenario für LCG-Generator, in dem a zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, a zu klein

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483647,a=-1,c=0,x0=1,n=45, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter a befindet sich nicht zwischen 0 und m.

### Test66\_ERR\_LcgBerechnung\_04\_A

Testet Fehlerszenario für LCG-Generator, in dem x0 zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, x0 zu gross

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=100,a=630360016,c=0,x0=100,n=501, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter a befindet sich nicht zwischen 0 und m.

### Test66\_ERR\_LcgBerechnung\_04\_B

Testet Fehlerszenario für LCG-Generator, in dem x0 zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, x0 zu klein

Ziel:Zufallsgenerierung

Generator:Linear-Kongruenz-Generator

LCG-Parameter:m=2147483647,a=630360016,c=0,x0=-1,n=51, divide=true

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter x0 befindet sich nicht zwischen 0 und m (inklusiv)

### Test67\_ERR\_BjarnscheZufallsmethode\_01\_A

Testet Fehlerszenario für Bjarnsche-Zufallsmethode, in dem n zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, n zu gross

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=20,x0=1,n=50001

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter n befindet sich nicht zwischen 0 und 50000 (inklusiv).

### Test67\_ERR\_BjarnscheZufallsmethode\_01\_B

Testet Fehlerszenario für Bjarnsche-Zufallsmethode, in dem n zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, n zu klein

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=20,x0=1,n=0

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter n befindet sich nicht zwischen 0 und 50000 (inklusiv).

### Test67\_ERR\_BjarnscheZufallsmethode\_02\_A

Testet Fehlerszenario für Bjarnsche-Zufallsmethode, in dem m zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, m zu gross

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=18446744073709551616,x0=1,n=100

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

Konvertierung von LCG-Parameter mit Wert <18446744073709551616> nicht möglich.

### Test67\_ERR\_BjarnscheZufallsmethode\_02\_B

Testet Fehlerszenario für Bjarnsche-Zufallsmethode, in dem m zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, m zu klein

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=0,x0=1,n=100

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter m befindet sich nicht zwischen 0 und 2^64 (long-range)

### Test67\_ERR\_BjarnscheZufallsmethode\_03\_A

Testet Fehlerszenario für Bjarnsche-Zufallsmethode, in dem x0 zu groß ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, x0 zu gross

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=100,x0=100,n=100

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter x0 befindet sich nicht zwischen 0 und m (inklusiv)

### Test67\_ERR\_BjarnscheZufallsmethode\_03\_B

Testet Fehlerszenario für Bjarnsche-Zufallsmethode, in dem x0 zu klein ist.

*Ausgabe:*

-----INPUT-----

# Teste Grenzwert, x0 zu klein

Ziel:Zufallsgenerierung

Generator:Bjarnsche-Zufallsmethode

Bjarnsche-Zufallsmethode-Parameter:m=100,x0=-1,n=100

-----OUTPUT-----

Fehlermeldung:

Fehler beim Berechnen - Berechnung konnte nicht durchgeführt werden.

Details:

LCG-Parameter x0 befindet sich nicht zwischen 0 und m (inklusiv)

# Interpretation der Ergebnisse

# Ausblick

Generell bietet die aktuelle Implementierung die Möglichkeit, sämtliche Verfahren, um weitere zu erweitern, da der Presenter für die Zufallzahlengeneratoren für beide Use-Case-Szenarien Interfaces besitzt. Diese lassen sich individuell durch weitere implementierende Klassen ergänzen.

Die Herausforderung besteht darin, dass der Presenter derzeit beim Initialisieren der Zufallsgenerator- und Bewertungsobjekte zwischen sämtlichen Instanzen unterscheiden muss, um die richtige zu nutzen. Erweitert man nun die Verfahrensarten, wird der Presenter durch die einzupflegenden Unterscheidungen stets komplexer.

Folglich ist es sinnvoll sich Gedanken über ein Pattern zu machen, welches die Initialisierungen von Objekten vereinfacht.

Somit wäre eine Möglichkeit zunächst ein abstraktes Klassenkonstrukt zu konstruieren, welches generelle Informationen zu den zu initialisierenden Parametern beinhaltet. Passen kann im Anschluss durch Vererbung zwischen verschiedenen Arten unterschieden werden.

Durch diese Unterscheidungsmöglichkeiten lässt sich eine Anbindung eines Factory-Patterns ermöglichen.

Dieses kümmert sich um die passende Initialisierung der Generatoren und Gütefunktionen in Abhängigkeit eines Vererbungsobjektes. Somit wäre die Initialisierung nicht mehr direkter Teil des Presenters und dieser müsste nur auf die Factory zugreifen.

# Interpretation der Ergebnisse

# Anwendung und Benutzeranleitung

Dieses Kapitel behandelt allgemeine Themen bezüglich der Anwendung sowie Dokumentation. Dazu zählen:

* Verwendete Hilfsmittel
* Ordnerstruktur
* Installation und Ausführung

## Verwendete Hilfsmittel

Folgende Hilfsmittel wurden für die Erstellung der Dokumentation und Software verwendet:

* Visual Paradigm für Diagramme und Modellierungen
* Structorizer für Nassi-Shneiderman-Diagramme
* Word 365 für die Dokumentation
* IntelliJ Ultimate 2019.3
* Github für Datensicherung

# Eidesstattliche Versicherung

*„Ich erkläre verbindlich, dass das vorliegende Prüfprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind in der Arbeit vollständig aufgeführt. Ich versichere, dass der vorgelegte Ausdruck mit dem Inhalt der von mir erstellten digitalen Version identisch ist. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung des Prüfprodukts als Prüfleistung ausschließt.“*

Ort, Datum, Unterschrift